**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Основы программирования»

Отчет по лабораторной работе №1

Выполнила: Кичикова Александра

Группа: ИУ5- 32Б

Москва, 2021 г.

### **Задание:**

Задание лабораторной работы состоит из решения нескольких задач.

Файлы, содержащие решения отдельных задач, должны располагаться в пакете lab\_python\_fp. Решение каждой задачи должно раполагаться в отдельном файле.

При запуске каждого файла выдаются тестовые результаты выполнения соответствующего задания.

### **Задача 1 (файл field.py)**

Необходимо реализовать генератор field. Генератор field последовательно выдает значения ключей словаря.

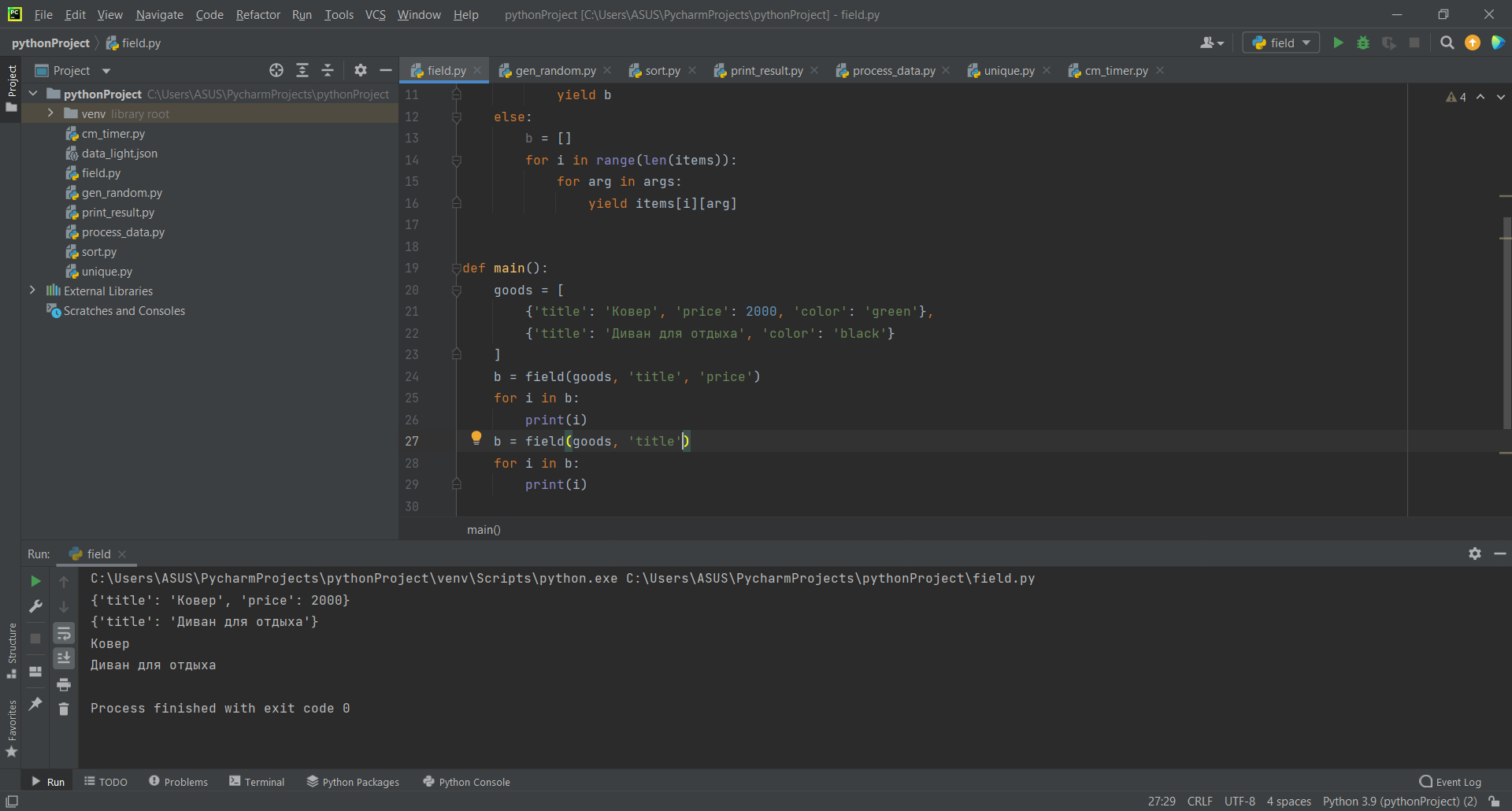
* В качестве первого аргумента генератор принимает список словарей, дальше через \*args генератор принимает неограниченное количествово аргументов.
* Если передан один аргумент, генератор последовательно выдает только значения полей, если значение поля равно None, то элемент пропускается.
* Если передано несколько аргументов, то последовательно выдаются словари, содержащие данные элементы. Если поле равно None, то оно пропускается. Если все поля содержат значения None, то пропускается элемент целиком.

Текст программы:

def field(items, \*args):  
 assert len(args) > 0  
 if len(args) > 1:  
 for i in range(len(items)):  
 b = {}  
 for arg in args:  
 try:  
 b[arg] = items[i][arg]  
 except:  
 pass  
 yield b  
 else:  
 b = []  
 for i in range(len(items)):  
 for arg in args:  
 yield items[i][arg]  
  
  
def main():  
 goods = [  
 {'title': 'Ковер', 'price': 2000, 'color': 'green'},  
 {'title': 'Диван для отдыха', 'color': 'black'}  
 ]

b = field(goods, 'title', 'price')  
 for i in b:  
 print(i)  
 b = field(goods, 'title')  
 for i in b:  
 print(i)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Экранная форма с примером выполнения программы:



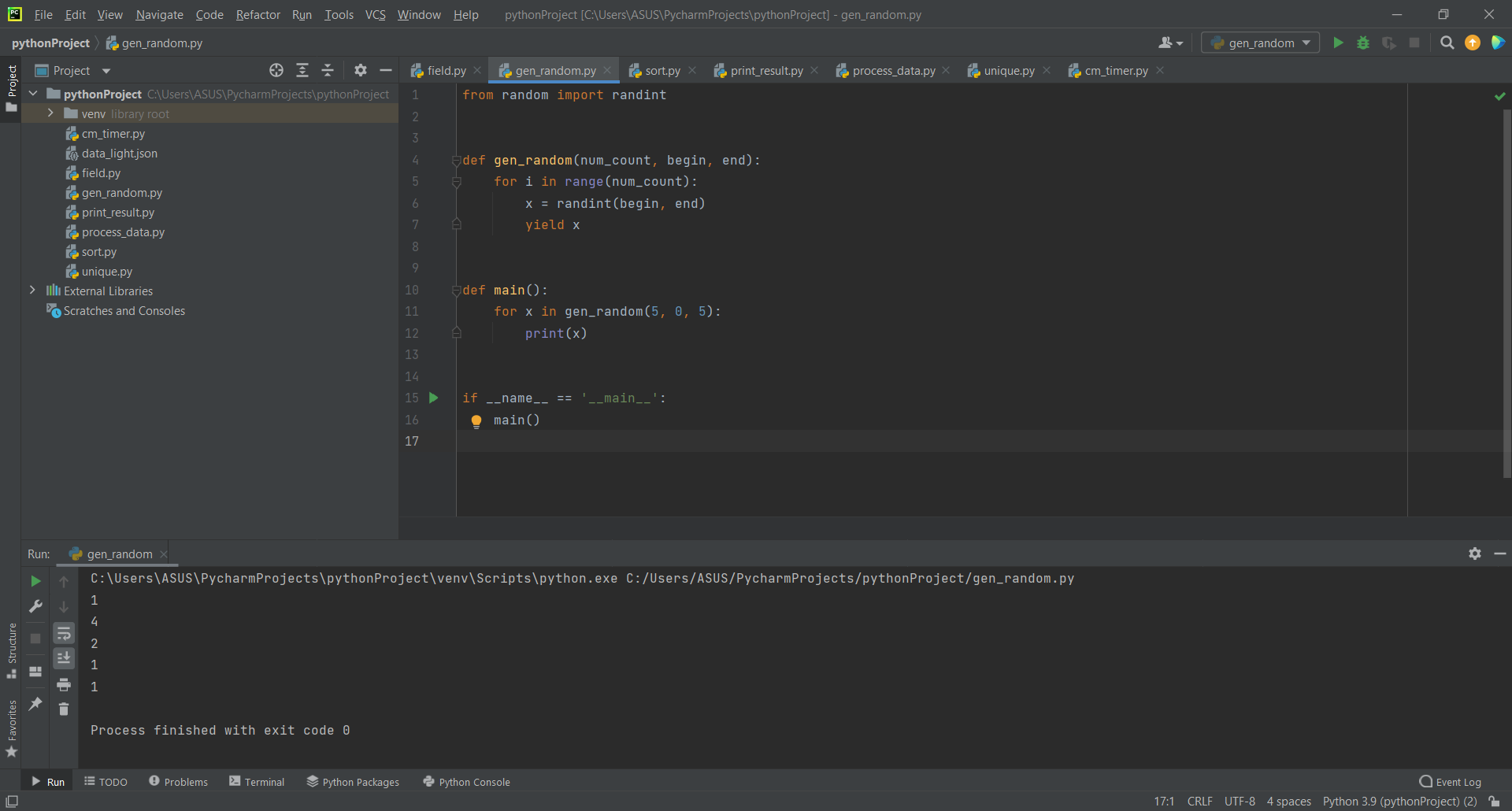
### **Задача 2 (файл gen\_random.py)**

Необходимо реализовать генератор gen\_random(количество, минимум, максимум), который последовательно выдает заданное количество случайных чисел в заданном диапазоне от минимума до максимума, включая границы диапазона.

Текст программы:

from random import randint  
  
  
def gen\_random(num\_count, begin, end):  
 for i in range(num\_count):  
 x = randint(begin, end)  
 yield x  
  
  
def main():  
 for x in gen\_random(5, 0, 5):  
 print(x)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Экранная форма с примером выполнения программы:



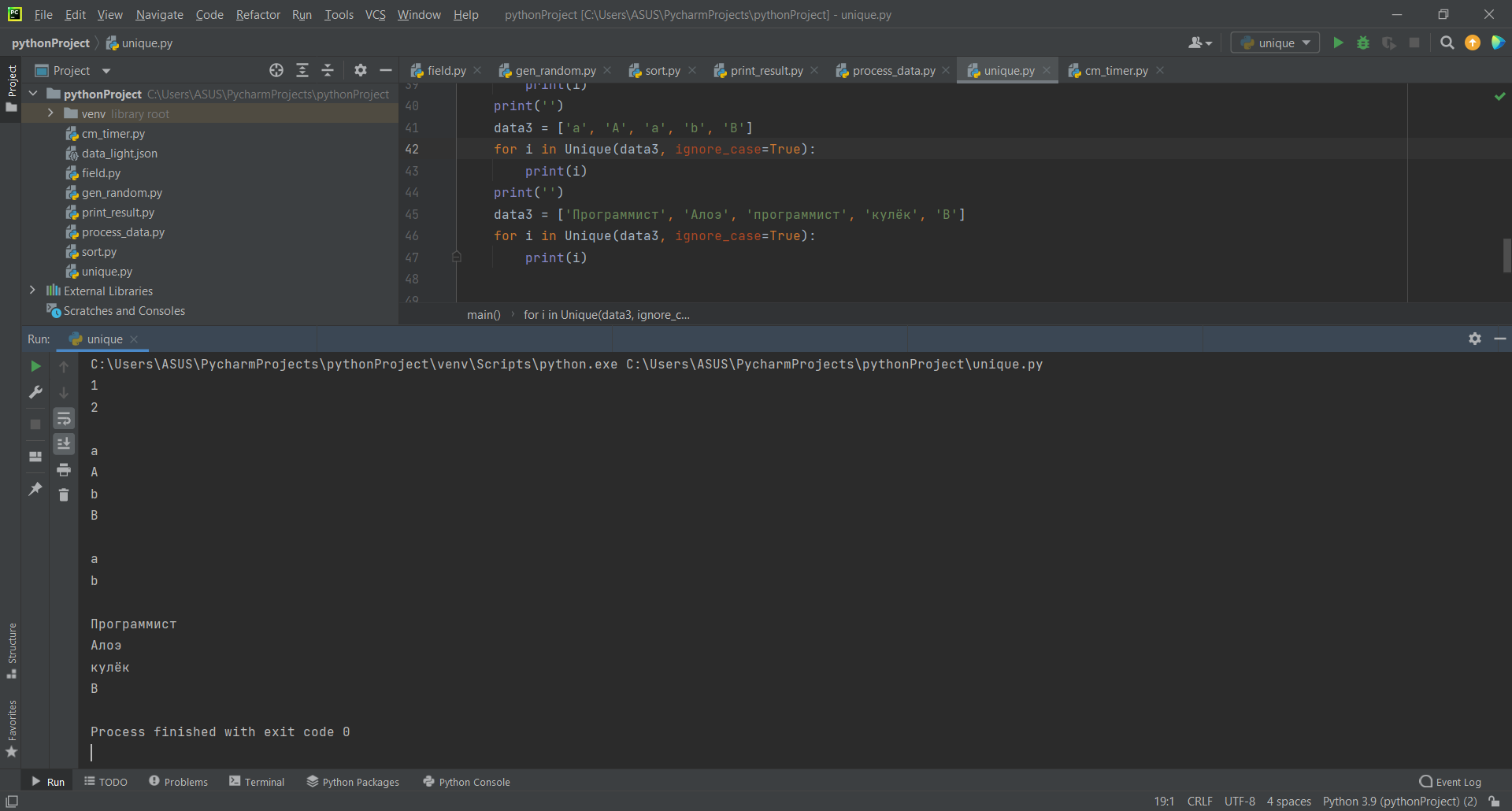
### **Задача 3 (файл unique.py)**

* Необходимо реализовать итератор Unique(данные), который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты.
* Конструктор итератора также принимает на вход именованный bool-параметр ignore\_case, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен False.
* При реализации необходимо использовать конструкцию \*\*kwargs.
* Итератор должен поддерживать работу как со списками, так и с генераторами.
* Итератор не должен модифицировать возвращаемые значения.

Текст программы:

class Unique(object):  
 def \_\_init\_\_(self, items, \*\*kwargs):  
 self.used\_elements = list()  
 self.data = []  
 for i in items:  
 self.data.append(i)  
 self.index = 0  
 self.case = len(kwargs)  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 while True:  
 if self.index >= len(self.data):  
 raise StopIteration  
 else:  
 if self.case == 0:  
 current = self.data[self.index]  
 self.index = self.index + 1  
 if current not in self.used\_elements:  
 self.used\_elements.append(current)  
 return current  
 else:  
 current = self.data[self.index]  
 self.index = self.index + 1  
 if current.lower() not in self.used\_elements:  
 self.used\_elements.append(current.lower())  
 return current  
  
  
def main():  
 data1 = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]  
 for i in Unique(data1):  
 print(i)  
 print('')  
 data2 = ['a', 'A', 'a', 'b', 'B']  
 for i in Unique(data2):  
 print(i)  
 print('')  
 data3 = ['a', 'A', 'a', 'b', 'B']  
 for i in Unique(data3, ignore\_case=True):  
 print(i)  
 print('')  
 data3 = ['Программист', 'Алоэ', 'программист', 'кулёк', 'B']  
 for i in Unique(data3, ignore\_case=True):  
 print(i)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Экранная форма с примером выполнения программы:



### **Задача 4 (файл sort.py)**

Дан массив 1, содержащий положительные и отрицательные числа. Необходимо **одной строкой кода** вывести на экран массив 2, которые содержит значения массива 1, отсортированные по модулю в порядке убывания. Сортировку необходимо осуществлять с помощью функции sorted.

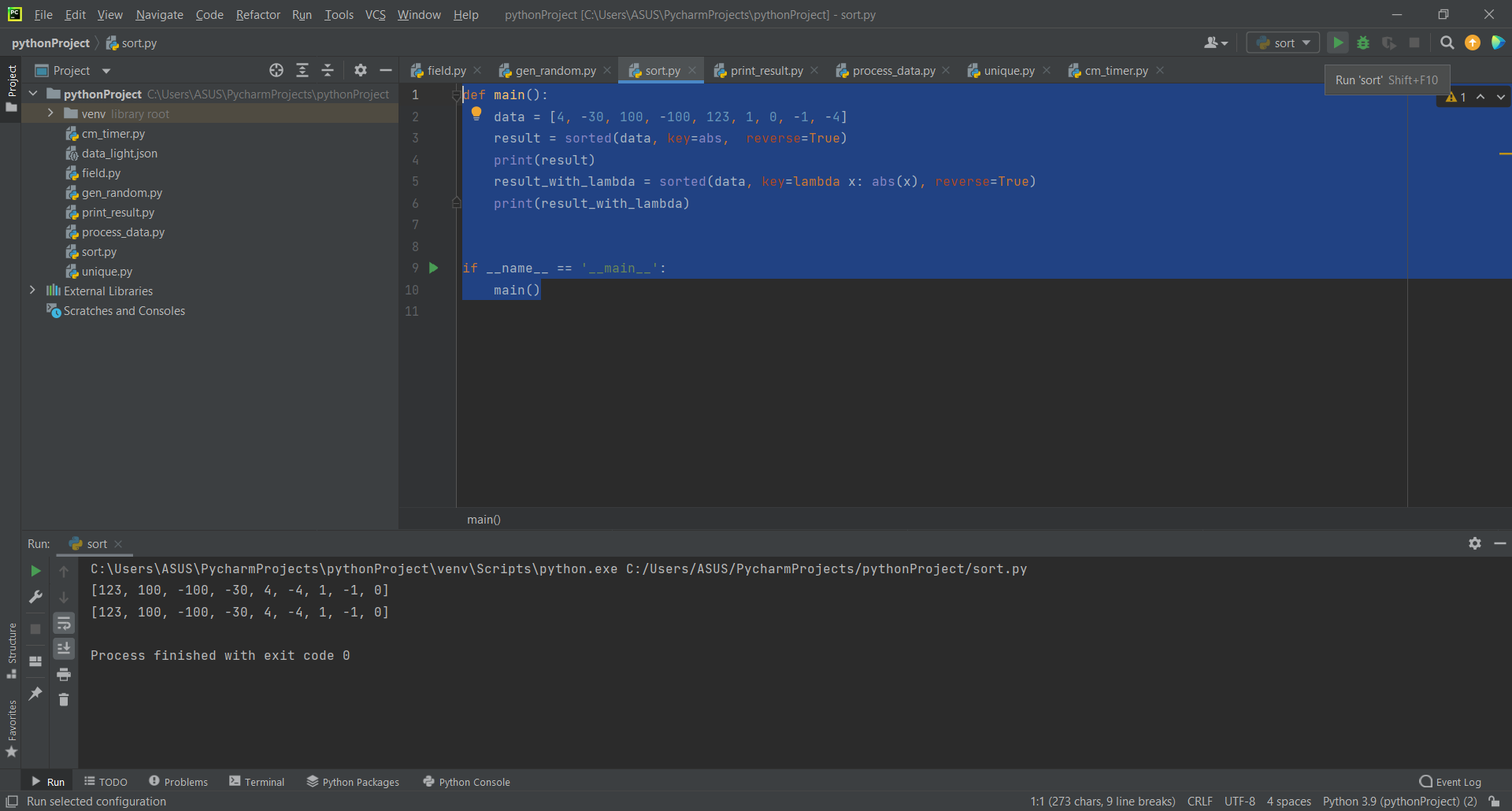
Необходимо решить задачу двумя способами:

1. С использованием lambda-функции.
2. Без использования lambda-функции.

Текст программы:

def main():  
 data = [4, -30, 100, -100, 123, 1, 0, -1, -4]  
 result = sorted(data, key=abs, reverse=True)  
 print(result)  
 result\_with\_lambda = sorted(data, key=lambda x: abs(x), reverse=True)  
 print(result\_with\_lambda)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Экранная форма с примером выполнения программы:



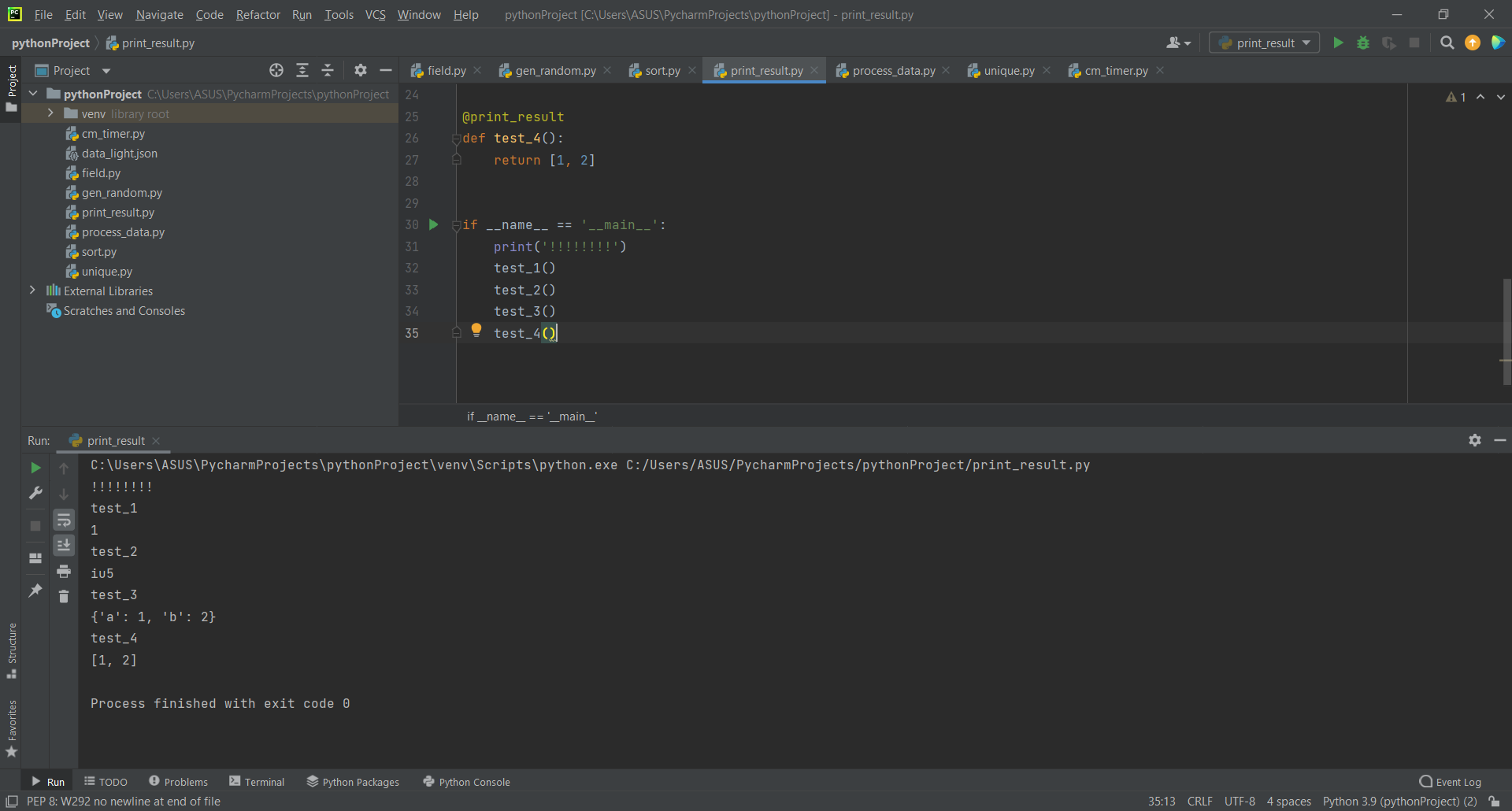
### **Задача 5 (файл print\_result.py)**

Необходимо реализовать декоратор print\_result, который выводит на экран результат выполнения функции.

Текст программы:

def print\_result(func\_to\_decorate):  
 def wrapper(\*args, \*\*kwargs):  
 print(func\_to\_decorate.\_\_name\_\_)  
 res = func\_to\_decorate(\*args, \*\*kwargs)  
 print(res)  
 return res  
 return wrapper  
  
  
@print\_result  
def test\_1():  
 return 1  
  
  
@print\_result  
def test\_2():  
 return 'iu5'  
  
  
@print\_result  
def test\_3():  
 return {'a': 1, 'b': 2}  
  
  
@print\_result  
def test\_4():  
 return [1, 2]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 test\_1()  
 test\_2()  
 test\_3()  
 test\_4()

Экранная форма с примером выполнения программы:



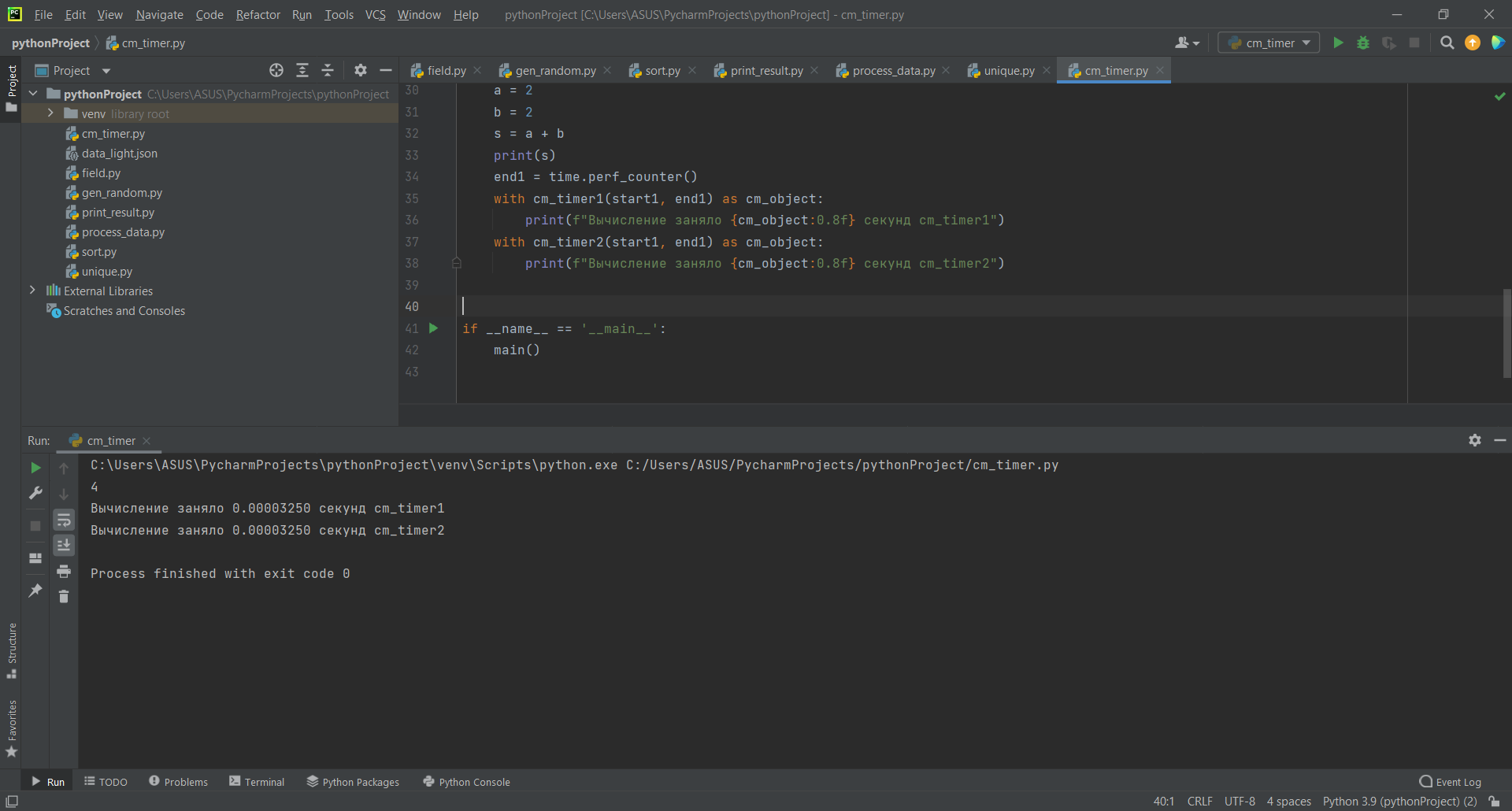
### **Задача 6 (файл cm\_timer.py)**

Необходимо написать контекстные менеджеры cm\_timer\_1 и cm\_timer\_2, которые считают время работы блока кода и выводят его на экран.

Текст программы:

import time  
from contextlib import contextmanager  
  
  
class cm\_timer1:  
  
 def \_\_init\_\_(self, start, end):  
 self.start = start  
 self.end = end  
  
 def \_\_enter\_\_(self):  
 self.elapsed\_time = self.end - self.start  
 return self.elapsed\_time  
  
 def \_\_exit\_\_(self, exp\_type, exp\_value, traceback):  
 if exp\_type is not None:  
 print(exp\_type, exp\_value, traceback)  
 else:  
 pass  
  
  
@contextmanager  
def cm\_timer2(start, end):  
 elapsed\_time = end - start  
 yield elapsed\_time  
  
  
def main():  
 start1 = time.perf\_counter()  
 a = 2  
 b = 2  
 s = a + b  
 print(s)  
 end1 = time.perf\_counter()  
 with cm\_timer1(start1, end1) as cm\_object:  
 print(f"Вычисление заняло {cm\_object:0.8f} секунд cm\_timer1")  
 with cm\_timer2(start1, end1) as cm\_object:  
 print(f"Вычисление заняло {cm\_object:0.8f} секунд cm\_timer2")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Экранная форма с примером выполнения программы:



### **Задача 7 (файл process\_data.py)**

* В предыдущих задачах были написаны все требуемые инструменты для работы с данными. Применим их на реальном примере.
* В файле [data\_light.json](https://github.com/ugapanyuk/BKIT_2021/blob/main/notebooks/fp/files/data_light.json) содержится фрагмент списка вакансий.
* Структура данных представляет собой список словарей с множеством полей: название работы, место, уровень зарплаты и т.д.
* Необходимо реализовать 4 функции - f1, f2, f3, f4. Каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора @print\_result печатается результат, а контекстный менеджер cm\_timer\_1 выводит время работы цепочки функций.
* Предполагается, что функции f1, f2, f3 будут реализованы в одну строку. В реализации функции f4 может быть до 3 строк.
* Функция f1 должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих задач.
* Функция f2 должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Для фильтрации используйте функцию filter.
* Функция f3 должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию map.
* Функция f4 должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте zip для обработки пары специальность — зарплата.

Текст программы:

import json  
from print\_result import print\_result  
from field import field  
from gen\_random import gen\_random  
from unique import Unique  
from cm\_timer import cm\_timer1  
import time  
import pathlib  
from pathlib import Path  
path = Path(pathlib.Path.home(), 'PycharmProjects', 'pythonProject', 'data\_light.json')  
  
with open(path, encoding="utf-8") as f:  
 data = json.load(f)  
  
  
@print\_result  
def f1(arg):  
 return list(Unique(field(arg, 'job-name'), ignore\_case=True))  
  
  
@print\_result  
def f2(arg):  
 return list(filter(lambda x: x.startswith('Программист'), arg))  
  
@print\_result  
def f3(arg):  
 return list(map(lambda x: x + ' с опытом Python', arg))  
  
  
@print\_result  
def f4(arg):  
 a = list(zip(arg, gen\_random(len(arg), 100000, 200000)))  
 return list(map(lambda x: x[0] + ' с зарплатой ' + str(x[1]), a))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 start1 = time.perf\_counter()  
 f4(f3(f2(f1(data))))  
 end1 = time.perf\_counter()  
 with cm\_timer1(start1, end1) as cm\_object:  
 print(f"Вычисление заняло {cm\_object:0.8f} секунд cm\_timer1")

Экранная форма с примером выполнения программы:

Ввиду того, что результат работы f1 слишком объемен, ниже он представлен не полностью.

